Socket编程UDP

UDP网络编程

V1版本 - Echo server

简单的回显服务器和客户端代码

备注: 代码中会用到 地址转换函数 . 参考接下来的章节.

nocopy.hpp

代码块

```
#pragma once
#include <iostream>
class nocopy
{
public:
    nocopy() {}
    nocopy(const nocopy &) = delete;
    const nocopy &operator=(const nocopy &) = delete;
    ~nocopy() {}
};
```

UdpServer.hpp

代码块

```
#pragma once
#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "nocopy.hpp"
#include "Log.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"
```

```cpp
const static uint16_t defaultport = 8888;
const static int defaultfd = -1;
const static int defaultsize = 1024;
class UdpServer : public nocopy
{
public:
    UdpServer(uint16_t port = defaultport)
        : _port(port), _sockfd(defaultfd)
    {
    }
    void Init()
    {
        // 1. 创建socket，就是创建了文件细节
        _sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (_sockfd < 0)
        {
            lg.LogMessage(Fatal, "socket errr, %d : %s\n", errno,
                          strerror(errno));
            exit(Socket_Err);
        }
        lg.LogMessage(Info, "socket success, sockfd: %d\n", _sockfd);
        // 2. 绑定，指定网络信息
        struct sockaddr_in local;
        bzero(&local, sizeof(local)); // memset
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = INADDR_ANY; // 0
        // local.sin_addr.s_addr = inet_addr(_ip.c_str()); // 1. 4字节IP 2. 变
        成网络序列
        // 结构体填完，设置到内核中了吗？？没有
        int n = ::bind(_sockfd, (struct sockaddr *)&local, sizeof(local));
        if (n != 0)
        {
            lg.LogMessage(Fatal, "bind errr, %d : %s\n", errno,
                          strerror(errno));
            exit(Bind_Err);
        }
    }
    void Start()
    {
        // 服务器永远不退出
        char buffer[defaultsize];
        for (;;)
        {
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer); // 不能乱写
            ssize_t n = recvfrom(_sockfd, buffer, sizeof(buffer) - 1, 0,
```

```
63                                      (struct sockaddr *)&peer, &len);
64              if (n > 0)
65              {
66                  InetAddr addr(peer);
67                  buffer[n] = 0;
68                  std::cout << "[" << addr.PrintDebug() << "]# " << buffer <<
    std::endl;
69                  sendto(_sockfd, buffer, strlen(buffer), 0, (struct sockaddr
    *)&peer, len);
70              }
71          }
72      }
73      ~UdpServer()
74      {
75      }
76
77  private:
78      // std::string _ip; // 后面要调整
79      uint16_t _port;
80      int _sockfd;
81  };
```

Comm.hpp

代码块

```
1  #pragma once
2  enum
3  {
4      Usage_Err = 1,
5      Socket_Err,
6      Bind_Err
7  };
```

· `Log.hpp` 已经有了，这里就不再复制粘贴了

· 云服务器不允许直接bind公有IP，我们也不推荐编写服务器的时候，bind明确的IP，推荐直接写成 `INADDR_ANY`

在网络编程中，当一个进程需要绑定一个网络端口以进行通信时，可以使用INADDR_ANY作为IP地址参数。这样做意味着该端口可以接受来自任何IP地址的连接请求，无论是本地主机还是远程主机。例如，如果服务器有多个网卡（每个网卡上有不同的IP地址），使用INADDR_ANY可以省去确定数据是从服务器上具体哪个网卡/IP地址上面获取的。

UdpClient.hpp

代码块

```
#include <iostream>
#include <cerrno>
#include <cstring>
#include <string>
#include <unistd.h>
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
void Usage(const std::string &process)
{
    std::cout << "Usage: " << process << " server_ip server_port" << std::endl;
}
// ./udp_client server_ip server_port
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        Usage(argv[0]);
        return 1;
    }
    std::string serverip = argv[1];
    uint16_t serverport = std::stoi(argv[2]);
    // 1. 创建socket
    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0)
    {
        std::cerr << "socket error: " << strerror(errno) << std::endl;
        return 2;
    }
    std::cout << "create socket success: " << sock << std::endl;
    // 2. client要不要进行bind? 一定要bind的! !
    // 但是，不需要显示bind，client会在首次发送数据的时候会自动进行bind
    // 为什么? server端的端口号，一定是众所周知，不可改变的，client 需要 port，bind随
    机端口.
        // 为什么? client会非常多.
        // client 需要bind，但是不需要显示bind，让本地OS自动随机bind，选择随机端口号
        // 2.1 填充一下server信息
        struct sockaddr_in server;
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(serverport);
    server.sin_addr.s_addr = inet_addr(serverip.c_str());
    while (true)
    {
        // 我们要发的数据
        std::string inbuffer;
```

```
48          std::cout << "Please Enter# ";
49          std::getline(std::cin, inbuffer);
50          // 我们要发给谁呀? server
51          ssize_t n = sendto(sock, inbuffer.c_str(), inbuffer.size(), 0, (struct
    sockaddr *)&server, sizeof(server));
52          if (n > 0)
53          {
54              char buffer[1024];
55              // 收消息
56              struct sockaddr_in temp;
57              socklen_t len = sizeof(temp);
58              ssize_t m = recvfrom(sock, buffer, sizeof(buffer) - 1, 0, (struct
    sockaddr *)&temp, &len); // 一般建议都是要填的.
59              if (m > 0)
60              {
61                  buffer[m] = 0;
62                  std::cout << "server echo# " << buffer << std::endl;
63              }
64              else
65                  break;
66          }
67          else
68              break;
69      }
70      close(sock);
71      return 0;
72  }
```

· client端要不要显示bind的问题

V2 版本 - DictServer

实现一个简单的英译汉的网络字典

dict.txt

代码块

```
1  apple: 苹果
2  1banana: 香蕉
3  cat: 猫
4  dog: 狗
5  book: 书
6  pen: 笔
7  happy: 快乐的
8  sad: 悲伤的
9  run: 跑
```

```
jump: 跳
teacher: 老师
student: 学生
car: 汽车
bus: 公交车
love: 爱
hate: 恨
hello: 你好
goodbye: 再见
summer: 夏天
winter: 冬天
```

Dict.hpp

代码块

```cpp
#pragma once
#include <iostream>
#include <string>
#include <fstream>
#include <unordered_map>
const std::string sep = ": ";
class Dict
{
private:
    void LoadDict()
    {
        std::ifstream in(_confpath);
        if (!in.is_open())
        {
            std::cerr << "open file error" << std::endl; // 后面可以用日志替代打
            印 return;
        }
        std::string line;
        while (std::getline(in, line))
        {
            if (line.empty())
                continue;
            auto pos = line.find(sep);
            if (pos == std::string::npos)
                continue;
            std::string key = line.substr(0, pos);
            std::string value = line.substr(pos + sep.size());
            _dict.insert(std::make_pair(key, value));
        }
        in.close();
```

```
    }

public:
    Dict(const std::string &confpath) : _confpath(confpath)
    {
        LoadDict();
    }
    std::string Translate(const std::string &key)
    {
        auto iter = _dict.find(key);
        if (iter == _dict.end())
            return std::string("Unknown");
        else
            return iter->second;
    }
    ~Dict()
    {
    }

private:
    std::string _confpath;
    std::unordered_map<std::string, std::string> _dict;
};
```

UdpServer.hpp

代码块

```
#pragma once
#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unordered_map>
#include <functional>
#include "nocopy.hpp"
#include "Log.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"
const static uint16_t defaultport = 8888;
```

```cpp
const static int defaultfd = -1;
const static int defaultsize = 1024;
using func_t = std::function<void(const std::string &req, std::string *resp)>;
class UdpServer : public nocopy
{
public:
    UdpServer(func_t func, uint16_t port = defaultport)
        : _func(func), _port(port), _sockfd(defaultfd)
    {
    }
    void Init()
    {
        // 1. 创建socket，就是创建了文件细节
        _sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (_sockfd < 0)
        {
            lg.LogMessage(Fatal, "socket errr, %d : %s\n", errno,
                          strerror(errno));
            exit(Socket_Err);
        }
        lg.LogMessage(Info, "socket success, sockfd: %d\n", _sockfd);
        // 2. 绑定，指定网络信息
        struct sockaddr_in local;
        bzero(&local, sizeof(local)); // memset
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = INADDR_ANY; // 0
        // local.sin_addr.s_addr = inet_addr(_ip.c_str()); // 1. 4字节IP 2. 变
        成网络序列
        // 结构体填完，设置到内核中了吗？？没有
        int n = ::bind(_sockfd, (struct sockaddr *)&local, sizeof(local));
        if (n != 0)
        {
            lg.LogMessage(Fatal, "bind errr, %d : %s\n", errno,
                          strerror(errno));
            exit(Bind_Err);
        }
    }
    void Start()
    {
        // 服务器永远不退出
        char buffer[defaultsize];
        for (;;)
        {
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer); // 不能乱写
            ssize_t n = recvfrom(_sockfd, buffer, sizeof(buffer) - 1, 0,
```

```
                                    (struct sockaddr *)&peer, &len);
            if (n > 0)
            {
                InetAddr addr(peer);
                buffer[n] = 0;
                std::cout << "[" << addr.PrintDebug() << "]# " << buffer << std::endl;
                std::string value;
                _func(buffer, &value); // 回调业务翻译方法
                sendto(_sockfd, value.c_str(), value.size(), 0, (struct sockaddr *)&peer, len);
            }
        }
    }
    ~UdpServer()
    {
    }

private:
    // std::string _ip; // 后面要调整
    uint16_t _port;
    int _sockfd;
    func_t _func;
};
```

Main.cc

代码块

```
#include "UdpServer.hpp"
#include "Comm.hpp"
#include "Dict.hpp"
#include <memory>
void Usage(std::string proc)
{
    std::cout << "Usage : \n\t" << proc << " local_port\n"
                << std::endl;
}
Dict gdict("./dict.txt");
void Execute(const std::string &req, std::string *resp)
{
    *resp = gdict.Translate(req);
}
// ./udp_server 8888
int main(int argc, char *argv[])
{
```

```
    if (argc != 2)
    {
        Usage(argv[0]);
        return Usage_Err;
    }
    // std::string ip = argv[1];
    uint16_t port = std::stoi(argv[1]);
    // Lambda写法
    // Dict gdict("./dict.txt");
    // std::unique_ptr<UdpServer> usvr = std::make_unique<UdpServer>(port,
    [&gdict](const std::string &message) -> std::string {
// return gdict.Translate(message);
//});
std::unique_ptr<UdpServer> usvr = std::make_unique<UdpServer>(Execute,
port);
usvr->Init();
usvr->Start();
return 0; }
```

V3 版本 - DictServer封装版

下面是一个封装版的，大家下来可以看一下

udp_socket.hpp

代码块

```
#pragma once
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cassert>
#include <string>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
typedef struct sockaddr sockaddr;
typedef struct sockaddr_in sockaddr_in;
class UdpSocket
{
public:
    UdpSocket() : fd_(-1)
    {
    }
    bool Socket()
    {
```

```
21          fd_ = socket(AF_INET, SOCK_DGRAM, 0);
22          if (fd_ < 0)
23          {
24              perror("socket");
25              return false;
26          }
27          return true;
28      }
29      bool Close()
30      {
31          close(fd_);
32          return true;
33      }
34      bool Bind(const std::string &ip, uint16_t port)
35      {
36          sockaddr_in addr;
37          addr.sin_family = AF_INET;
38          addr.sin_addr.s_addr = inet_addr(ip.c_str());
39          addr.sin_port = htons(port);
40          int ret = bind(fd_, (sockaddr *)&addr, sizeof(addr));
41          if (ret < 0)
42          {
43              perror("bind");
44              return false;
45          }
46          return true;
47      }
48      bool RecvFrom(std::string *buf, std::string *ip = NULL, uint16_t *port =
    NULL)
49      {
50          char tmp[1024 * 10] = {0};
51          sockaddr_in peer;
52          socklen_t len = sizeof(peer);
53          ssize_t read_size = recvfrom(fd_, tmp,
54                                       sizeof(tmp) - 1, 0, (sockaddr *)&peer,
    &len);
55          if (read_size < 0)
56          {
57              perror("recvfrom");
58              return false;
59          }
60          // 将读到的缓冲区内容放到输出参数中
61          buf->assign(tmp, read_size);
62          if (ip != NULL)
63          {
64              *ip = inet_ntoa(peer.sin_addr);
65          }
```

```
66          if (port != NULL)
67          {
68              *port = ntohs(peer.sin_port);
69          }
70          return true;
71      }
72      bool SendTo(const std::string &buf, const std::string &ip, uint16_t port)
73      {
74          sockaddr_in addr;
75          addr.sin_family = AF_INET;
76          addr.sin_addr.s_addr = inet_addr(ip.c_str());
77          addr.sin_port = htons(port);
78          ssize_t write_size = sendto(fd_, buf.data(), buf.size(), 0,
79                                      (sockaddr *)&addr, sizeof(addr));
80          if (write_size < 0)
81          {
82              perror("sendto");
83              return false;
84          }
85          return true;
86      }
87
88  private:
89      int fd_;
90  };
```

UDP通用服务器

udp_server.hpp

代码块

```
1   #pragma once
2   #include "udp_socket.hpp"
3   // C 式写法
4   // typedef void (*Handler)(const std::string& req, std::string* resp);
5   // C++ 11 式写法, 能够兼容函数指针, 仿函数, 和 lambda
6   #include <functional>
7   typedef std::function<void(const std::string &, std::string *resp)> Handler;
8   class UdpServer
9   {
10  public:
11      UdpServer()
12      {
13          assert(sock_.Socket());
14      }
15      ~UdpServer()
```

```
16      {
17          sock_.Close();
18      }
19      bool Start(const std::string &ip, uint16_t port, Handler handler)
20      {
21          // 1. 创建 socket
22          // 2. 绑定端口号
23          bool ret = sock_.Bind(ip, port);
24          if (!ret)
25          {
26              return false;
27          }
28          // 3. 进入事件循环
29          for (;;)
30          {
31              // 4. 尝试读取请求
32              std::string req;
33              std::string remote_ip;
34              uint16_t remote_port = 0;
35              bool ret = sock_.RecvFrom(&req, &remote_ip, &remote_port);
36              if (!ret)
37              {
38                  continue;
39              }
40              std::string resp;
41              // 5. 根据请求计算响应
42              handler(req, &resp);
43              // 6. 返回响应给客户端
44              sock_.SendTo(resp, remote_ip, remote_port);
45              printf("[%s:%d] req: %s, resp: %s\n", remote_ip.c_str(),
    remote_port,
46                     req.c_str(), resp.c_str());
47          }
48          sock_.Close();
49          return true;
50      }
51
52  private:
53      UdpSocket sock_;
54  };
```

实现英译汉服务器

以上代码是对 udp 服务器进行通用接口的封装. 基于以上封装, 实现一个查字典的服务器就很容易了.

dict_server.cc

代码块

```
#include "udp_server.hpp"
#include <unordered_map>
#include <iostream>
std::unordered_map<std::string, std::string> g_dict;
void Translate(const std::string &req, std::string *resp)
{
    auto it = g_dict.find(req);
    if (it == g_dict.end())
    {
        *resp = "未查到!";
        return;
    }
    *resp = it->second;
}
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        printf("Usage ./dict_server [ip] [port]\n");
        return 1;
    }
    // 1. 数据初始化
    g_dict.insert(std::make_pair("hello", "你好"));
    g_dict.insert(std::make_pair("world", "世界"));
    g_dict.insert(std::make_pair("c++", "最好的编程语言"));
    g_dict.insert(std::make_pair("bit", "特别NB"));
    // 2. 启动服务器
    UdpServer server;
    server.Start(argv[1], atoi(argv[2]), Translate);
    return 0;
}
```

UDP通用客户端

udp_client.hpp

代码块

```
#pragma once 1
#include "udp_socket.hpp"
class UdpClient
{
public:
    UdpClient(const std::string &ip, uint16_t port) : ip_(ip), port_(port)
    {
        assert(sock_.Socket());
    }
```

```
    ~UdpClient()
    {
        sock_.Close();
    }
    bool RecvFrom(std::string *buf)
    {
        return sock_.RecvFrom(buf);
    }
    bool SendTo(const std::string &buf)
    {
        return sock_.SendTo(buf, ip_, port_);
    }

private:
    UdpSocket sock_;
    // 服务器端的 IP 和 端口号
    std::string ip_;
    uint16_t port_;
};
```

实现英译汉客户端

代码块

```
#include "udp_client.hpp"
#include <iostream>
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        printf("Usage ./dict_client [ip] [port]\n");
        return 1;
    }
    UdpClient client(argv[1], atoi(argv[2]));
    for (;;)
    {
        std::string word;
        std::cout << "请输入您要查的单词: ";
        std::cin >> word;
        if (!std::cin)
        {
            std::cout << "Good Bye" << std::endl;
            break;
        }
        client.SendTo(word);
        std::string result;
```

```
23          client.RecvFrom(&result);
24          std::cout << word << " 意思是 " << result << std::endl;
25      }
26      return 0;
27  }
```

V4版本 - 简单聊天室

Route.hpp

代码块

```
1   #pragma once
2   #include <iostream>
3   #include <string>
4   #include <vector>
5   #include "InetAddr.hpp"
6   #include "Log.hpp"
7   using namespace LogModule;
8   class Route
9   {
10  private:
11      bool IsExist(InetAddr &peer)
12      {
13          for (auto &user : _online_user)
14          {
15              if (user == peer)
16              {
17                  return true;
18              }
19          }
20          return false;
21      }
22      void AddUser(InetAddr &peer)
23      {
24          LOG(LogLevel::INFO) << "新增一个在线用户: " << peer.StringAddr();
25          _online_user.push_back(peer);
26      }
27      void DeleteUser(InetAddr &peer)
28      {
29          for (auto iter = _online_user.begin(); iter != _online_user.end();
30               iter++)
31          {
32              if (*iter == peer)
33              {
34                  LOG(LogLevel::INFO) << "删除一个在线用户:" << peer.StringAddr()
```

```
                                        << "成功";
                _online_user.erase(iter);
                break;
            }
        }
    }

public:
    Route()
    {
    }
    void MessageRoute(int sockfd, const std::string &message, InetAddr &peer)
    {
        if (!IsExist(peer))
        {
            AddUser(peer);
        }
        std::string send_message = peer.StringAddr() + "# " + message; //127.0.0.1 : 8080 #你好
            // TODO
            for (auto &user : _online_user)
        {
            sendto(sockfd, send_message.c_str(), send_message.size(), 0, (const struct sockaddr *)&(user.NetAddr()), sizeof(user.NetAddr()));
        }
        // 这个用户一定已经在线了
        if (message == "QUIT")
        {
            LOG(LogLevel::INFO) << "删除一个在线用户: " << peer.StringAddr();
            DeleteUser(peer);
        }
    }
    ~Route()
    {
    }

private:
    // 首次给我发消息，等同于登录
    std::vector<InetAddr> _online_user; // 在线用户
};
```

UdpServer.hpp

代码块

```
#pragma once
```

```cpp
#include <iostream>
#include <string>
#include <functional>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "Log.hpp"
#include "InetAddr.hpp"
using namespace LogModule;
using func_t = std::function<void(int sockfd, const std::string &, InetAddr &)>;
const int defaultfd = -1;
// 你是为了进行网络通信的!
class UdpServer
{
public:
    UdpServer(uint16_t port, func_t func)
        : _sockfd(defaultfd),
          // _ip(ip),
          _port(port),
          _isrunning(false),
          _func(func)
    {
    }
    void Init()
    {
        // 1. 创建套接字
        _sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (_sockfd < 0)
        {
            LOG(LogLevel::FATAL) << "socket error!";
            exit(1);
        }
        LOG(LogLevel::INFO) << "socket success, sockfd : " << _sockfd;
        // 2. 绑定socket信息，ip和端口， ip(比较特殊，后续解释)
        // 2.1 填充sockaddr_in结构体
        struct sockaddr_in local;
        bzero(&local, sizeof(local));
        local.sin_family = AF_INET;
        // 我会不会把我的IP地址和端口号发送给对方?
        // IP信息和端口信息，一定要发送到网络!
        // 本地格式->网络序列
        local.sin_port = htons(_port);
        // IP也是如此，1. IP转成4字节 2. 4字节转成网络序列 -> in_addr_t
        inet_addr(const char *cp);
```

```
48          // local.sin_addr.s_addr = inet_addr(_ip.c_str()); // TODO
49          local.sin_addr.s_addr = INADDR_ANY;
50          // 那么为什么服务器端要显式的bind呢? IP和端口必须是众所周知且不能轻易改变的!
51          int n = bind(_sockfd, (struct sockaddr *)&local, sizeof(local));
52          if (n < 0)
53          {
54              LOG(LogLevel::FATAL) << "bind error";
55              exit(2);
56          }
57          LOG(LogLevel::INFO) << "bind success, sockfd : " << _sockfd;
58      }
59      void Start()
60      {
61          _isrunning = true;
62          while (_isrunning)
63          {
64              char buffer[1024];
65              struct sockaddr_in peer;
66              socklen_t len = sizeof(peer);
67              // 1. 收消息, client为什么要个服务器发送消息啊? 不就是让服务端处理数据。
68              ssize_t s = recvfrom(_sockfd, buffer, sizeof(buffer) - 1, 0,
69                                   (struct sockaddr *)&peer, &len);
70              if (s > 0)
71              {
72                  InetAddr client(peer);
73                  buffer[s] = 0;
74                  // TODO
75                  _func(_sockfd, buffer, client);
76                  // LOG(LogLevel::DEBUG) << "[" << peer_ip << ":" <<
77                  peer_port << "]# " << buffer; // 1. 消息内容 2. 谁发的? ?
78                  // 2. 发消息
79                  // std::string echo_string = "server echo@ ";
80                  // echo_string += buffer;
81                  // sendto(_sockfd, result.c_str(), result.size(), 0, (struct
    sockaddr*)&peer, len);
82              }
83          }
84      }
85      ~UdpServer()
86      {
87      }
88
89  private:
90      int _sockfd;
91      uint16_t _port;
92      // std::string _ip; // 用的是字符串风格, 点分十进制, "192.168.1.1"
93      bool _isrunning;
```

```
94        func_t _func; // 服务器的回调函数，用来进行对数据进行处理
95    };
```

· 引入线程池，这里就不重复贴代码了

InetAddr.hpp

代码块

```
1   #pragma once
2   #include <iostream>
3   #include <string>
4   #include <sys/types.h>
5   #include <sys/socket.h>
6   #include <netinet/in.h>
7   #include <arpa/inet.h>
8   class InetAddr
9   {
10  public:
11      InetAddr(struct sockaddr_in &addr) : _addr(addr)
12      {
13          _port = ntohs(_addr.sin_port);
14          _ip = inet_ntoa(_addr.sin_addr);
15      }
16      std::string Ip() { return _ip; }
17      uint16_t Port() { return _port; };
18      std::string PrintDebug()
19      {
20          std::string info = _ip;
21          info += ":";
22          info += std::to_string(_port); // "127.0.0.1:4444"
23          return info;
24      }
25      const struct sockaddr_in &GetAddr()
26      {
27          return _addr;
28      }
29      bool operator==(const InetAddr &addr)
30      {
31          // other code
32          return this->_ip == addr._ip && this->_port == addr._port;
33      }
34      ~InetAddr() {}
35
36  private:
37      std::string _ip;
38      uint16_t _port;
```

```
39      struct sockaddr_in _addr;
40  };
```

· 在InetAddr中，重载一下 == 方便对用户是否是同一个进行比较

ServerMain.cc

代码块

```
1   // 单进程服务器
2   // std::unique_ptr<UdpServer> usvr = std::make_unique<UdpServer>(port, [&r]
3   (int sockfd, const std::string &message, InetAddr &peer)
4   {
5       // r.MessageRoute(sockfd, message, peer);
6       // });
7       // 进程池服务器
8       // 1. 路由服务
9       std::unique_ptr<Route> r = std::make_unique<Route>();
10      // 2. 线程池
11      auto tp = ThreadPool<task_t>::GetInstance();
12      // 3. 网络服务器对象，提供通信功能
13      std::unique_ptr<UdpServer> usvr = std::make_unique<UdpServer>(port, [&r,
14                                                                    &tp]
    (int sockfd, const std::string &message, InetAddr &peer)
15                                                                        {
16  task_t t = std::bind(&Route::MessageRoute, r.get(), sockfd, message,
17  peer);
18  tp->Enqueue(t); });
```

UdpClient.hpp

代码块

```
1   #include <iostream>
2   #include <cerrno>
3   #include <cstring>
4   #include <string>
5   #include <unistd.h>
6   #include <sys/types.h> /* See NOTES */
7   #include <sys/socket.h>
8   #include <arpa/inet.h>
9   #include <netinet/in.h>
10  #include "Thread.hpp"
11  #include "InetAddr.hpp"
12  void Usage(const std::string &process)
13  {
```

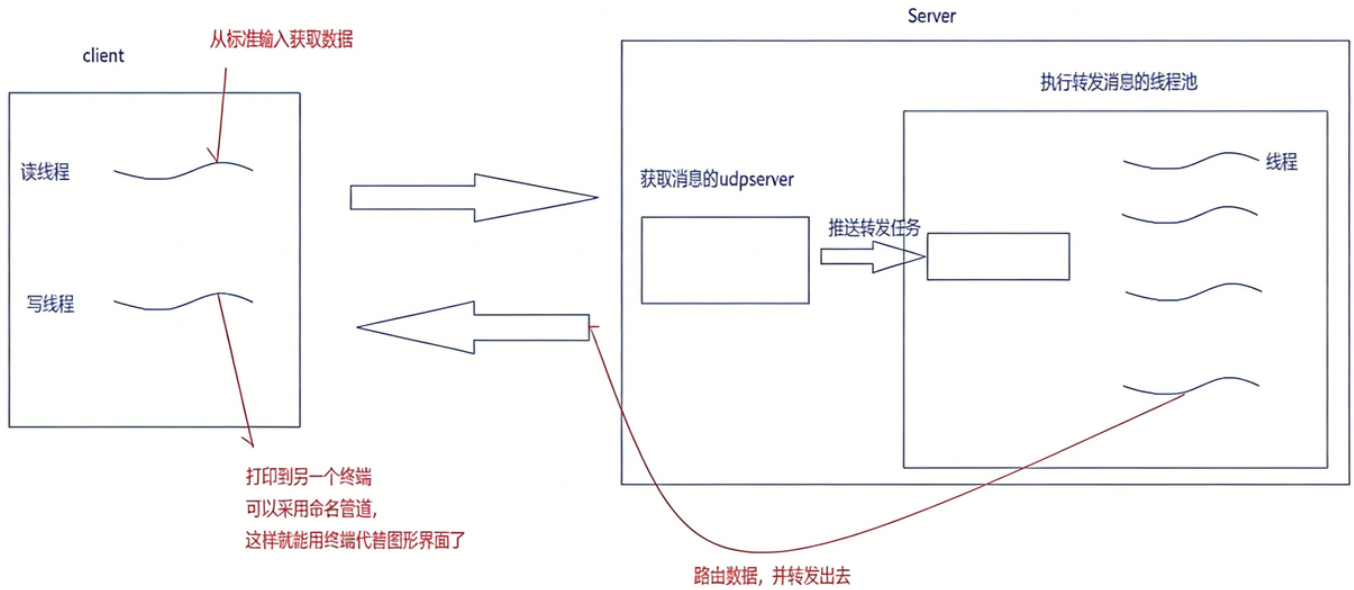
```
14      std::cout << "Usage: " << process << " server_ip server_port" << std::endl;
15  }
16  class ThreadData
17  {
18  public:
19      ThreadData(int sock, struct sockaddr_in &server) : _sockfd(sock),
20                                                         _serveraddr(server)
21      {
22      }
23      ~ThreadData()
24      {
25      }
26
27  public:
28      int _sockfd;
29      InetAddr _serveraddr;
30  };
31  void RecverRoutine(ThreadData &td)
32  {
33      char buffer[4096];
34      while (true)
35      {
36          struct sockaddr_in temp;
37          socklen_t len = sizeof(temp);
38          ssize_t n = recvfrom(td._sockfd, buffer, sizeof(buffer) - 1, 0,
39                               (struct sockaddr *)&temp, &len); // 一般建议都是要填
    的.
40          if (n > 0)
41          {
42              buffer[n] = 0;
43              std::cerr << buffer << std::endl; // 方便一会查看效果
44          }
45          else
46              break;
47      }
48  }
49  // 该线程只负责发消息
50  void SenderRoutine(ThreadData &td)
51  {
52      while (true)
53      {
54          // 我们要发的数据
55          std::string inbuffer;
56          std::cout << "Please Enter# ";
57          std::getline(std::cin, inbuffer);
58          auto server = td._serveraddr.GetAddr();
59          // 我们要发给谁呀? server
```

```
60          ssize_t n = sendto(td._sockfd, inbuffer.c_str(), inbuffer.size(), 0,
61                             (struct sockaddr *)&server, sizeof(server));
62          if (n <= 0)
63              std::cout << "send error" << std::endl;
64      }
65  }
66  // ./udp_client server_ip server_port
67  int main(int argc, char *argv[])
68  {
69      if (argc != 3)
70      {
71          Usage(argv[0]);
72          return 1;
73      }
74      std::string serverip = argv[1];
75      uint16_t serverport = std::stoi(argv[2]);
76      // 1. 创建socket
77      // udp是全双工的。既可以读，也可以写，可以同时读写，不会多线程读写的问题
78      int sock = socket(AF_INET, SOCK_DGRAM, 0);
79      if (sock < 0)
80      {
81          std::cerr << "socket error: " << strerror(errno) << std::endl;
82          return 2;
83      }
84      std::cout << "create socket success: " << sock << std::endl;
85      // 2. client要不要进行bind? 一定要bind的！！但是，不需要显示bind，client会在首次发送数据的时候会自动进行bind
86      // 为什么? server端的端口号，一定是众所周知，不可改变的，client 需要 port，bind随机端口.
87      // 为什么? client会非常多.
88      // client 需要bind，但是不需要显示bind，让本地OS自动随机bind，选择随机端口号
89      // 2.1 填充一下server信息
90      struct sockaddr_in server;
91      memset(&server, 0, sizeof(server));
92      server.sin_family = AF_INET;
93      server.sin_port = htons(serverport);
94      server.sin_addr.s_addr = inet_addr(serverip.c_str());
95      ThreadData td(sock, server);
96      Thread<ThreadData> recver("recver", RecverRoutine, td);
97      Thread<ThreadData> sender("sender", SenderRoutine, td);
98      recver.Start();
99      sender.Start();
100     recver.Join();
101     sender.Join();
102     close(sock);
103     return 0;
104 }
```

· UDP协议支持全双工，一个sockfd，既可以读取，又可以写入，对于客户端和服务端同样如此

· 多线程客户端，同时读取和写入

· 测试的时候，使用管道进行演示



补充参考内容

地址转换函数

本节只介绍基于IPv4的socket网络编程,sockaddr_in中的成员struct in_addr sin_addr表示32位 的IP地址

但是我们通常用点分十进制的字符串表示IP 地址,以下函数可以在字符串表示 和in_addr表示之间转换;

字符串转in_addr的函数:

```
#include <arpa/inet.h>

int inet_aton(const char *strptr,
struct in_addr *addrptr);

in_addr_t inet_addr(const char
*strptr);
int inet_pton(int family, const char
*strptr, void *addrptr);
```

in_addr转字符串的函数:

```
char *inet_ntoa(struct in_addr inaddr);

const char *inet_ntop(int family,
const void *addrptr, char *strptr,
size_t len);
```

其中inet_pton和inet_ntop不仅可以转换IPv4的in_addr,还可以转换IPv6的in6_addr,因此函数接口是void *addrptr。

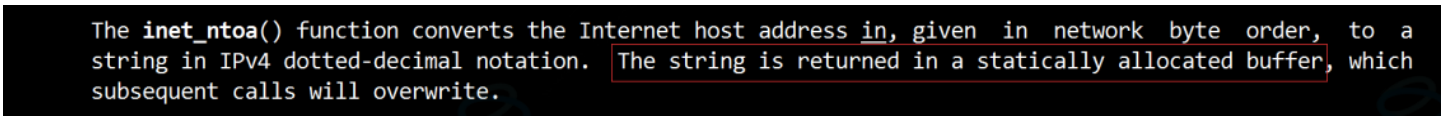
代码示例:

```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main() {
  struct sockaddr_in addr;
  inet_aton("127.0.0.1", &addr.sin_addr);
  uint32_t* ptr = (uint32_t*)(&addr.sin_addr);
  printf("addr: %x\n", *ptr);
  printf("addr_str: %s\n", inet_ntoa(addr.sin_addr));
  return 0;
}
```

关于inet_ntoa

inet_ntoa这个函数返回了一个char*, 很显然是这个函数自己在内部为我们申请了一块内存来保存ip的结果. 那么是否需要调用者手动释放呢?



man手册上说, inet_ntoa函数, 是把这个返回结果放到了静态存储区. 这个时候不需要我们手动进行释放.那么问题来了, 如果我们调用多次这个函数, 会有什么样的效果呢? 参见如下代码:

```c
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main() {
  struct sockaddr_in addr1;
  struct sockaddr_in addr2;
  addr1.sin_addr.s_addr = 0;
  addr2.sin_addr.s_addr = 0xffffffff;
  char* ptr1 = inet_ntoa(addr1.sin_addr);
  char* ptr2 = inet_ntoa(addr2.sin_addr);
  printf("ptr1: %s, ptr2: %s\n", ptr1, ptr2);
  return 0;
}
```

运行结果如下:

```
[tangzhong@tz addr_convert]$ ./a.out
ptr1: 255.255.255.255, ptr2: 255.255.255.255
```

因为inet_ntoa把结果放到自己内部的一个静态存储区, 这样第二次调用时的结果会覆盖掉上一次的结果.

> · 思考: 如果有多个线程调用 inet_ntoa, 是否会出现异常情况呢?
>
> · 在APUE中, 明确提出inet_ntoa不是线程安全的函数;
>
> · 但是在centos7上测试, 并没有出现问题, 可能内部的实现加了互斥锁;
>
> · 同学们课后自己写程序验证一下在自己的机器上inet_ntoa是否会出现多线程的问题;
>
> · 在多线程环境下, 推荐使用inet_ntop, 这个函数由调用者提供一个缓冲区保存结果, 可以规避线程安全问题;

多线程调用inet_ntoa代码示例如下

代码块

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
void *Func1(void *p)
{
    struct sockaddr_in *addr = (struct sockaddr_in *)p;
    while (1)
    {
```

```
        char *ptr = inet_ntoa(addr->sin_addr);
        printf("addr1: %s\n", ptr);
    }
    return NULL;
}
void *Func2(void *p)
{
    struct sockaddr_in *addr = (struct sockaddr_in *)p;
    while (1)
    {
        char *ptr = inet_ntoa(addr->sin_addr);
        printf("addr2: %s\n", ptr);
    }
    return NULL;
}
int main()
{
    pthread_t tid1 = 0;
    struct sockaddr_in addr1;
    struct sockaddr_in addr2;
    addr1.sin_addr.s_addr = 0;
    addr2.sin_addr.s_addr = 0xffffffff;
    pthread_create(&tid1, NULL, Func1, &addr1);
    pthread_t tid2 = 0;
    pthread_create(&tid2, NULL, Func2, &addr2);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    return 0;
}
```

remove_if 样例

代码块

```
#include <iostream>
#include <list>
#include <memory>
#include <algorithm>
int main()
{
    std::list<std::shared_ptr<int>> ls;
    ls.push_back(std::make_shared<int>(1));
    ls.push_back(std::make_shared<int>(2));
    ls.push_back(std::make_shared<int>(3));
    ls.push_back(std::make_shared<int>(4));
    ls.push_back(std::make_shared<int>(4));
```

```
    ls.push_back(std::make_shared<int>(4));
    ls.push_back(std::make_shared<int>(5));
    ls.push_back(std::make_shared<int>(6));
    for (auto &v : ls)
    {
        std::cout << *v << std::endl;
    }
    std::cout << "aa: " << ls.size() << std::endl;
    std::cout << "\n";
    // int a = 3;
    int a = 4;
    auto pos = remove_if(ls.begin(), ls.end(), [&a](const std::shared_ptr<int> &elem) -> bool
                         { return a == *elem; });
    ls.erase(pos, ls.end());
    std::cout << "aa: " << ls.size() << std::endl;
    for (auto &v : ls)
    {
        std::cout << *v << std::endl;
    }
    return 0;
}

//remove_if()并不会实际移除序列[start, end)中的元素；如果在一个容器上应用remove_if()，容器的长度并不会改变，所有的元素都还在容器里面（但是逻辑上已经无法访问）.remove_if()将所有应该移除的元素都移动到容器尾部并返回一个分界的迭代器. 为了实际移除元素，你必须对容器自行调用erase()以擦除需要移除的元素
```